Transformação MediESTecaEst-v1EF ➔ MediESTecaEst -v2Ident (Com Microsoft Identity)

Remover e/ou comentar o código anterior que irá ser alterado

1. Remover os controladores e as vistas para os **Utentes**, **Livros** e **Requisições**
2. Comentar as opções de menu para as ações destes controladores no ficheiro **_layout.cshtml**
3. Alterar o nome da classe **Utente** para **UtenteOld** para permitir criar uma nova classe **Utente**.

Adicionar **Microsoft Identity**

4. No projeto adicionar **new ScaffoldItem** e escolher **Identity**
    a. Na janela de inserção, escolher a página de *layout* existente
        ~/Views/Shared/_Layout.cshtml
    b. Escolher agora o **override** dos ficheiros
        Account\Register e **Account\Manage\index**
    c. Adicionar um novo contexto:
        MediatecaEst.Data.**MediESTecaComIdentity**Context
    d. Adicionar como **user class** a (nova) classe **Utente**
5. Tendo em conta que foi criada uma nova classe **`Utente`** pela **Identity** na pasta **Areas/Identity/Data** vamos passar a utilizar esta classe. Sendo assim:
    a. Copiar os atributos **`Nome`** e **`Telefone`** juntamente com as suas anotações para a nova classe, renomear a classe **UtenteOld** para **Utente** e apagar o ficheiro da classe renomeada. Compile para garantir que a nova classe Utente está a ser usada sem erros (poderá ter que referir o novo *namespace* da nova classe Utente).
    b. Na classe **`Requisicao`** incluir a utilização da nova classe e alterar o tipo da propriedade **`UtenteId`** de inteiro para string, uma vez que é este tipo que é usado nas chaves primárias da tabela dos utilizadores criada pela Identity.
6. Efetuar as seguintes alterações que seguem as instruções da Microsoft para o Scaffold da Identity em projetos MVC sem este pacote:
    a. Adicionar a vista _LoginPartial ao menu do layout da aplicação logo debaixo da linha `</ul>`

```
<partial name="_LoginPartial" />
```

    b. No ficheiro **Program.cs**, comentar a criação do contexto que estava a ser usado na aplicação:

```
// builder.Services.AddDbContext<MediESTecaContext>(options =>
//
options.UseSqlServer(builder.Configuration.GetConnectionString("MediESTecaConte
xt")));
```

    O contexto que irá ser usado passa então a ser:

```
var connectionString =

builder.Configuration.GetConnectionString("MediESTecaComIdentityContextConnection");
```

```
builder.Services.AddDbContext<MediESTecaComIdentityContext>(options =>
    options.UseSqlServer(connectionString));
```

Copiar o código de *seed* dos livros da classe de contexto **`MediESTecaContext`** para a nova classe de contexto respeitante à criação dos livros para a nova classe de contexto em **Areas/Identity/Data**.

Elimine o ficheiro com a classe **`MediESTecaContext`** que deixa de ser utilizada.

Depois adiciona-se o serviço da Microsoft Identity (caso não tenha já sido criado):

```
builder.Services.AddDefaultIdentity<Utente>(options =>
                        options.SignIn.RequireConfirmedAccount = true)
                    .AddRoles<IdentityRole>()
                    .AddEntityFrameworkStores<MediESTecaComIdentityContext>();
```

c. No mesmo ficheiro, adicionar **`app.UseAuthentication();`** antes de **`app.UseAuthorization();`** se não tiver sido já adicionado. Incluir ainda depois do encaminhamento, **`app.MapRazorPages();`**

d. Fazer a nova classe de contexto derivar de **`IdentityContext<Utente>`**

7. **Verifique o resto de código**, deve compilar sem erros.

8. No ficheiro **Program.cs** alterar a adição do serviço Identity para:

```
builder.Services.AddDefaultIdentity<Utente>(options =>
       {
       // Password settings
       options.Password.RequireDigit = false;
       options.Password.RequiredLength = 6;
       options.Password.RequireNonAlphanumeric = false;
       options.Password.RequireUppercase = false;
       options.Password.RequireLowercase = false;

       options.SignIn.RequireConfirmedAccount = false;
       })
      .AddRoles<IdentityRole>()
      .AddEntityFrameworkStores<MediESTecaComIdentityContext>();
```

Passar a **utilizar o novo contexto** e a nova base de dados.

9. Eliminar a base de dados anterior utilizando a janela **Sql Server Object**

10. Eliminar a pasta das migrações, se esta tiver sido criada.

11. No **Package manager Console** executar os comandos **add-migration** e **update-database**.

Recriar parte do código anterior que foi eliminado.

12. Gerar os controladores e as vistas para os **Livros** e **Requisições** usando a nova classe de contexto.

13. Descomentar as opções de menu para as ações destes controladores no ficheiro **_layout.cshtml**

14. Atualizar a bases de dados e correr a aplicação para confirmar que está a funcionar

Adaptar a nova aplicação à utilização da nova classe **Utente** que deriva de **IdentityUser**

15. A gestão de utilizadores e dos papeis é feita agora através dos serviços existentes. Nesta aplicação, por uma questão de simplicidade, vamos optar por criar a informação inicial dos utilizadores e papeis da base de dados dentro do método **OnModelCreating** da classe **MediESTecaComIdentityContext**. Sendo assim, usamos o seguinte código:

```
PasswordHasher<Utente> ph = new PasswordHasher<Utente>();

Utente admin = new Utente
{
    UserName = "admin@ips.pt",
    NormalizedUserName = "admin@ips.pt".ToUpper(),
    Email = "admin@ips.pt",
    NormalizedEmail = "admin@ips.pt".ToUpper(),
    TwoFactorEnabled = false,
    EmailConfirmed = true,
    PhoneNumber = "123456789",
    PhoneNumberConfirmed = false,

    Nome = "admin",
    Telefone = 123456789
};
admin.PasswordHash = ph.HashPassword(admin, "123456");


Utente joao = new Utente
{
    UserName = "jonas@ip.pt",
    NormalizedUserName = "jonas@ip.pt".ToUpper(),
    Email = "jonas@ip.pt",
    NormalizedEmail = "jonas@ip.pt".ToUpper(),
    TwoFactorEnabled = false,
    EmailConfirmed = true,
    PhoneNumber = "123456789",
    PhoneNumberConfirmed = false,

    Nome = "João Manuel",
    Telefone = 1234567890
};
joao.PasswordHash = ph.HashPassword(joao, "123456");

Utente ana = new Utente
{
    UserName = "ana@gmail.com",
    NormalizedUserName = "ana@gmail.com".ToUpper(),
    Email = "ana@gmail.com",
    NormalizedEmail = "ana@gmail.com".ToUpper(),
    TwoFactorEnabled = false,
    EmailConfirmed = true,
    PhoneNumber = "234567890",
    PhoneNumberConfirmed = false,

    Nome = "Ana",
    Telefone = 1234567890
};
ana.PasswordHash = ph.HashPassword(ana, "123456");


builder.Entity<Utente>().HasData(
    admin,
    joao,
    ana
);
```

```
IdentityRole adminsRole = new IdentityRole { Name = "admins",
                                             NormalizedName = "admins".ToUpper() };
IdentityRole usersRole = new IdentityRole { Name = "users",
                                            NormalizedName = "users".ToUpper() };

builder.Entity<IdentityRole>().HasData(
   adminsRole,
   usersRole
);

builder.Entity<IdentityUserRole<string>>().HasData(
    new IdentityUserRole<string> { RoleId = adminsRole.Id, UserId = admin.Id },
    new IdentityUserRole<string> { RoleId = usersRole.Id, UserId = joao.Id },
    new IdentityUserRole<string> { RoleId = usersRole.Id, UserId = ana.Id }

);
```

16. Criar agora um controlador para os utentes usando um **empty Controller**, criar uma vista para a ação **Index** que liste o nome e o email dos utentes. Descomentar a ação no menu da aplicação em **_Layout.cshtml.**

17. Para reiniciar todos os dados, eliminar a base de dados e a pasta das migrações e voltar a criar a migração inicial e a base de dados.

18. Uma vez que temos uma classe de utilizador personalizada, vão se seguir as instruções da Microsoft para esta personalização. Fazer as correções aos ficheiros **Account\Register** e **Account\Manage\index** para que se recebam os campos do nome e telefone e estejam adaptados à classe **Utente**.

Incluir agora o **serviço de email** que é usado na página de registo

19. Não se irá utilizar ainda um serviço de email funcional. Ir-se-á apenas fornecer a classe de email sem nenhuma funcionalidade. Para isso, criar uma pasta **Services** e dentro dessa pasta definir a seguinte classe:

```
public class EmailSender : IEmailSender
{
    public Task SendEmailAsync(string email, string subject, string message)
    {
        return Task.CompletedTask;
    }
}
```

20. Adicionar agora o serviço de email no ficheiro **Program.cs**:

```
builder.Services.AddSingleton<IEmailSender, EmailSender>();
```

Criar um gestor de utilizadores para a nova aplicação e **restringir o acesso** a algumas páginas e funcionalidades dependendo do papel do utilizador

21. Criar o controlador **GestaoUtentes** para gerir os utilizadores – listar, editar e apagar utilizadores (com base na nova classe de modelo **UtenteEditViewModel**). Apenas o administrador terá acesso a este controlador, anotação na classe: **[Authorize(Roles = "admins")]**. Este controlador recebe o **UserManager** por dependency injection.

22. Para que não aparecesse no menu do site as entradas "Requisições" e "Utentes" para utilizadores que não tivessem feito o login, na página **_layout.cshtml** é feita a seguinte verificação para essas opções: **@if (User.Identity.IsAuthenticated)**. Também a entrada **"Gestão Utentes"** ficou limitada apenas aos administradores com a verificação: **@if User.IsInRole("admins"))**.

    a) Nota: Se o utilizador fizer o logout quando está numa das opções anteriores é redirecionado para a página onde está, neste caso para uma das ações dos controladores **Utentes**, **Requisicoes** ou **GestaoUtentes** às quais deixa de ter acesso por não estar "logado" e o browser passa a apresentar um erro. Sendo assim, será necessário na página **logout.cshtml.cs** da **Identity** trocar o retorno de **return LocalRedirect(returnUrl)** para **return LocalRedirect("/Home/index").**
    O processo é descrito em: https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-6.0#client-side-validation

22. Foi introduzida uma validação do lado do cliente que depende de dados na BD. Neste caso, não se pretendia adicionar um livro que tivesse um ISBN que já existisse para evitar a duplicação do ISBN do livro. Sendo assim, adicionou-se a anotação **[Remote(action: "IsbnExists", controller: "Livros")]** à propriedade **Isbn** da classe **Livro**. A validação é feita no método **IsbnExists** do controlador **Livros**.

    a) Nota: No método **IsbnExists** o nome do parâmetro deve ser igual ao que está definido na view. Neste caso, **isbn**, para que seja feito o binding corretamente e a variável receba o valor do código que se está a tentar atribuir.

23. No controlador dos Utentes foi alterado o construtor para que recebesse o serviço **UserManager**. Na ação **Index** deste controlador passou a usar-se o novo serviço para obter a lista de utentes que vai ser visualizada.

Novas restrições da aplicação

24. Limitar o acesso à lista de utentes aos utilizadores autenticados. Anotar o controlador respetivo com**[Authorize].** No *layout*, limitar o acesso à vista dos utentes e das requisições a quem está autenticado.

25. Limitar o acesso à criação, edição e remoção de livros ao administrador. Anotar com `[Authorize(Roles = "admins")].` Anotar as ações de **Index** e **Details** com `[AllowAnonymous]`. Na vista **Index** não mostrar os links de criação, edição ou remoção de Livros se o utilizador não estiver no papel de administrador.

26. Limitar o acesso às requisições a utilizadores autenticados. Não mostrar os links de edição se as requisições não forem do utilizador autenticado, não sendo este o administrador. Anotar o controlador com **[Authorize].** Na vista **Index** não mostrar os links de edição, detalhes ou remoção de Requisições se o utilizador não estiver no papel de administrador ou se não for quem fez a requisição.